First-Orderized ResearchCyc : Expressivity and Efficiency in a Common-Sense Ontology

Deepak Ramachandran[1] **Pace Reagan**[2] **and** **Keith Goolsbey**[2]
[1]Computer Science Department, University of Illinois at Urbana-Champaign, Urbana, IL 61801
dramacha@uiuc.edu
[2]Cycorp Inc., 3721 Executive Center Drive, Suite 100, Austin, TX 78731
{pace,goolsbey}@cyc.com

Abstract

Cyc is the largest existing common-sense knowledge base. Its ontology makes heavy use of higher-order logic constructs such as a context system, first class predicates, etc. Many of these higher-order constructs are believed to be key to Cyc's ability to represent common-sense knowledge and reason with it efficiently. In this paper, we present a translation of a large part (around 90%) of the Cyc ontology into First-Order Logic. We discuss our methodology, and the tradeoffs between expressivity and efficiency in representation and reasoning. We also present the results of experiments using VAMPIRE, SPASS, and the E Theorem Prover on the first-orderized Cyc KB. Our results indicate that, while the use of higher-order logic is not essential to the representability of common-sense knowledge, it greatly improves the efficiency of reasoning.

1 Introduction

ResearchCyc$^{\text{TM}}$ is a version of Cycorp Inc.'s Cyc®[1] Knowledge Base - the world's largest general common-sense ontology and reasoning engine. ResearchCyc is available under a free license, and consists of 1,074,484 assertions in a language of 122,658 symbols (not including strings or numbers). By contrast, the IEEE Suggested Upper Merged Ontology (Niles & Pease 2001) consists of 60,000 axioms over 20,000 terms.

A significant feature of ResearchCyc's design is the incorporation of higher-order assertions in its KB. (For the rest of this paper we will use the term ”higher-order” to mean any feature beyond ordinary first-order logic, like a context system or quantification over predicates.) The reasons for this are both philosophical and pragmatic; it is widely believed that a complete specification of what is understood to be "common-sense knowledge" requires some kind of higher-order features. For example, Boolos (Boolos 1984) discusses two sentences that cannot be represented in a logic without predicate quantification (*non-firstorderizable*):

1. Some critics admire only one another.
2. Some of Fianchetto's men went into the warehouse unaccompanied by anyone else.

The second sentence is non-firstorderizable if we consider "anyone else" to mean anyone who was not in the group of Fianchetto's men who went into the warehouse.

Another reason for the use of these higher-order constructs is the ability to represent knowledge succinctly and in a form that can be taken advantage of by specialized reasoning methods. The validation of this claim is one of the contributions of this paper.

In this paper, we present a tool for translating a significant percentage (around 90%) of the ResearchCyc KB into First-Order Logic (a process we call *FOLification*). Our methods involve a number of non-trivial transformations of the higher-order constructs appearing in ResearchCyc into FOL. In some cases (e.g. axiom schemata), the assertions could be translated precisely without any loss of semantics, but with an increase in the size of the representation (both the number of symbols and axioms). In other cases (e.g. contexts) only an approximate translation could be made. We describe our techniques in detail and the tradeoffs involved in expressivity and efficiency. Our final FOLified ResearchCyc KB consisted of 1,253,117 axioms over 132,116 symbols (not including strings or numbers).

We also present the preliminary results of some experiments performed on the FOLified KB using VAMPIRE (Riazanov 2003), SPASS (Weidenbach 1999), and the E theorem prover (Schulz 2002) with a sample collection of common-sense queries typical for ResearchCyc. The ATP systems performed orders of magnitude more slowly than the ResearchCyc inference engine that is custom built for Cyc's ontology and thus able to reason with its higher-order representations directly. From these results, we conclude that the types of problems Cyc is designed to handle differ substantially from the types of problems that the ATP systems are designed to handle. To our knowledge, this is the largest ever set of axioms upon which first-order theorem proving has been attempted. Our results are open to interpretation and we discuss them in the conclusion.

The rest of this paper is organized as follows: In Section 2 we provide some background. In Section 3 we discuss our FOLification procedure. In Section 4 we discuss the design of the CYC inference engine. Section 5 presents our preliminary experimental results. Finally in Section 6 we discuss conclusions and future work.

[1] Cyc is a registered trademark and ResearchCyc and OpenCyc are trademarks of Cycorp, Inc.

2 Background

We assume that the reader is familiar with all the standard terminology from the fields of Knowledge Representation and Automated Reasoning.

The Cyc common-sense knowledge base has been in development since 1984, first as a project at MCC, and since 1994 by Cycorp Inc. (*www.cyc.com*). A description of the motivation and goals of the Cyc project is given in (Lenat & Guha 1990). The complete Cyc knowledge base is under a proprietary license, but recently Cycorp has released a significant proportion of the KB to the academic community through the ResearchCyc project (*research.cyc.com*). ResearchCyc is available under a restricted but free license to all research institutions and universities. The eventual goal is to make almost all of the Cyc KB, except for certain sensitive sections, available in ResearchCyc. OpenCyc$^{\text{TM}}$(*www.opencyc.org*) is the open-source version of a subset of the Cyc KB (mostly the upper ontology) freely distributed under the Creative Commons license. In the rest of this paper, for brevity we will refer to all three KBs simply as Cyc, except where we wish to distinguish among them.

CycL is the declarative knowledge representation language used to store Cyc's KB. It has a Lisp-like syntax, with each atom and each term appearing within a pair of parentheses. As mentioned before, it is higher-order and has all the standard logical connectives and operators. The Cyc assertions and queries we use in the rest of this paper are written in simplified CycL.

TPTP (Thousands of Problems for Theorem Provers) (Sutcliffe & Suttner 1998) is the most popular library of test problems for automated theorem proving (ATP) systems and is the basis for the CADE ATP System Competition (Pelletier, Sutcliffe, & Suttner 2002). Our FOLification tool translates CycL into the TPTP format. We chose this format because many theorem provers accept it as input, and there exist tools for converting TPTP files into virtually every other first-order format.

3 First-Orderization of Higher-Order Constructs

In this section, we describe some of the transformations used to convert Cyc's higher-order KB into FOL.

It should be noted that our use of the term "higher-order" is rather loose; it signifies any conceivable extension to the usual syntactic restrictions of FOL: variable-arity predicates, sentences as arguments, etc.

3.1 Contexts

Contexts have been viewed both as a scheme for acheiving generality (McCarthy 1986) and locality (Giunchiglia 1993). ResearchCyc's context system enables both.

Each assertion in the KB is associated with a *microtheory* ("mt") that determines its context. The microtheories are arranged in a hierarchy, with each microtheory having access to all the assertions in its parents.

Microtheories are implemented by extending the language with the modal operator `ist(`k, φ`)`("is true") to express that the assertion φ is true in the context k. For example, consider the context of world mythology, in which we assert the fact that vampires fear garlic:

```
ist(WorldMythologyMt, (Vampire(X) ∧
Garlic(Y)) ⇒ (feelsTowardsObject ?X ?Y
Fear positiveAmountOf))
```

CycL has a predicate `genlMt` to assert that a microtheory is a child of another in the hierarchy (and thus inherits all its assertions):

```
genlMt(WorldMythologyMt, HumanActivitiesMt)
```

The closest analogue to Cyc's microtheory system in the literature is Buvac's quantificational theory of context (Buvac 1996).

There are two methods for handling microtheories in pure first-order logic. One is to treat each context as a separate module, choose at inference time which modules are required, and compile them into a single flat KB. This is the approach used by the Suggested Upper Merged Ontology (Niles & Pease 2001).

A more sophisticated scheme is to approximate the `ist` modality with an `mtVisible` predicate. For example,

```
mtVisible(WorldMythologyMt) ⇒ ((Vampire(X) ∧ Garlic(Y)) ⇒ (feelsTowardsObject ?X ?Y Fear positiveAmountOf))
```

Contexts can then inherit from each other:

```
mtVisible(WorldMythologyMt) ⇒ mtVisible(HumanActivitiesMt)
```

Criteria can be imposed for selecting certain contexts:

$$\forall k[(\texttt{mtVisible}(k) \Rightarrow \exists X \texttt{Dragon}(X)) \\ \Rightarrow \texttt{FictionalContext}(k)]$$

Both techniques mentioned above have one limitation: the inability to represent inter-contextual reasoning. In the module-based method there are no means of referring to other contexts. In the `mtVisible` method, while it is possible for assertions to refer to other contexts, the meaning is usually not what is intended. For example, suppose context k_1 knows that axiom ϕ is true in context k_2. If we try to translate this as:

$$\texttt{mtVisible(}k_1\texttt{)} \Rightarrow \texttt{(mtVisible(}k_2\texttt{)} \Rightarrow \phi\texttt{)}$$

This is equivalent to the statement:

$$\texttt{mtVisible(}k_2\texttt{)} \Rightarrow \texttt{(mtVisible(}k_1\texttt{)} \Rightarrow \phi\texttt{)}$$

which was not intended at all.

Our translation tool uses a combination of both methods. In general, each assertion is prefixed with the `mtVisible` condition for the microtheory it belongs to. But in the case of some microtheories being universally or almost universally visible, we use the modular approach. For example, when the assertion is from either UniversalVocabularyMt or BaseKB, then it is FOLified without adding the mtVisible predicate at all, since assertions from these microtheories are visible to practically every query.

3.2 Ontology

The backbone of Cyc's ontology of types and objects is the *genls* hierarchy. It is analogous to the genlMt hierarchy of microtheories. Every Thing in Cyc is either an Individual or a Collection. Both Individuals and Collections can be instances of Collections. Instancehood is declared using the isa predicate:

```
(isa Fido Dog)
(isa Dog Collection)
```

Collections can be declared to be subcollections, or "specs" of other collections using the genls predicate as a subclassing relation as follows:

```
(genls Fruit PlantPart)
```

which means that every instance of Fruit is also an instance of PlantPart.

The FOLifier creates a unique unary predicate for each collection, e.g. Fruit(x), PlantPart(x). This has the potential to make resolution-style inference easier because there is less choice of literals to resolve on due to greater variety in the predicate position. Furthermore, it is assumed that the heuristics of theorem provers are generally optimized for type hierarchies being expressed as unary predicates.

Given this, the genls assertion shown earlier can simply be translated as:

$$\forall x[\texttt{Fruit}(x) \Rightarrow \texttt{PlantPart}(x)]$$

Exceptions to the above technique are the *reified collection-denoting terms*, e.g. (JuvenileFn Human) which represents the collection of all young humans. (Note that a collection-denoting function need not necessarily denote a subcollection of its argument, for example (FruitFn AppleTree) does not denote a subcollection of AppleTree.) In our FOLification, we ignore the functional nature of the term and simply create a new predicate by concatenating the names of the function and its argument(s).

3.3 Predicate and Function Quantification

Allowing predicates to be arguments to other predicates and functions often enables more compact representations in the Cyc KB.

For example the meta-predicate TransitivePred(x) can be defined as follows:

$$\texttt{TransitivePred(P)} \Rightarrow$$
$$\forall x \forall y \forall z \texttt{P}(x,y) \wedge \texttt{P}(y,z) \Rightarrow \texttt{P}(x,z)$$

A more complicated example is

```
relationAllExists(pred, type1, type2)
```

which states that for every instance of type1, x, there must be some instance of type2, y s.t. pred(x,y). For example relationAllExists(has,Dog,Nose) states that every dog has a nose.

In ResearchCyc terminology, pseudo-higher-order constructs such as the above are called *rule macro predicates* and it is usually straightforward to convert these into FOL by replacing the literal with a first-order axiom schema, as in the TransitivePred example.

Another case in which predicate/function quantification can be translated into FOL is with argIsa assertions that are used to constrain the value of arguments to a predicate. For example,

```
argIsa(performedBy, 1, Action)
```

states that the first argument of a performedBy assertion must be an instance of Action. This can be translated into FOL as follows:

$$\forall x \forall y[\texttt{performedBy}(x,y) \Rightarrow \texttt{Action}(x)]$$

3.4 Variable Arity Predicates and Functions

It is natural to regard many predicates and functions as having variable arities, e.g. TheSet which extensionally enumerates a set. In some cases, the arities can be bounded by a small number, e.g. the Cyc function Meter can take either 1 or 2 arguments. (Meter 1) denotes "one meter", and (Meter 1 2) denotes "between one and two meters". In those cases, we can translate into FOL by suffixing the name of the predicate or function with the arity with which it appears in the assertion i.e.

In many cases, for example TheSet, the arity is fundamentally unbounded. In these cases, no straightforward translation seems to exist. An exception is the different predicate which, when applied to n arguments, can be translated into $O(n^2)$ inequality relations.

3.5 Exceptions

Exceptions to assertions in Cyc can be declared using the exceptFor and exceptWhen predicates, e.g.

```
 (exceptFor Taiwan-RepublicOfChina
(implies (isa ?X ChineseProvince)
(geopoliticalSubdivision China-
PeoplesRepublic ?X)))
```

Exception assertions like the above are used to give Cyc non-monotonic behavior.

When translating an exception assertion into FOL, the FOLifier could take the exception conditions and add them

FOL untranslatability reason	# of Assertions
UNBOUNDED ARITY FUNCTION	77756
META-SENTENCE	54209
META-VARIABLE	18923
VARIABLE ARITY FUNCTION	14187
FUNCTION ARG CONSTRAINT	7481
HOOK	6407
COLLECTION QUANTIFICATION	2263
UNBOUNDED ARITY PREDICATE	1554
FUNCTION QUANTIFICATION	1072
PREDICATE QUANTIFICATION	766
ILL-FORMED	458
INTER-MT	295
UNEXPECTED	247
SEQUENCE-VAR	206
NON-COLLECTION	132
NON-FUNCTION	114
NON-PREDICATE	62
VARIABLE ARITY PREDICATE	0

Table 1: ResearchCyc Translation Statistics

as negated literals in the antecedent of the excepted rule. In the general case this is more difficult to handle. Also, there can be problems when the assertion and the exception appear in different microtheories.

Currently, our FOLifier does not handle exceptions.

3.6 Translation Statistics

Table 1 gives a breakdown of how each assertion in the ResearchCyc KB was handled by our FOLifier. Overall 960,327 (out of 1,074,484) assertions were translated successfully. For the others, we list the reasons why a translation was not possible and the number of assertions for which that reason applies.

UNBOUNDED ARITY PREDICATE and UNBOUNDED ARITY FUNCTION indicate that the assertion has a predicate or function with variable arity, and that there is no upper bound on the maximum arity, as described in Section 3.4.

VARIABLE ARITY PREDICATE and VARIABLE ARITY FUNCTION indicate that the assertion has a predicate or function with variable arity, but there is an upper bound on the maximum arity. Many of these arise from Cyc's representation of scalar intervals. These are in principle translatable as described in Section 3.4, but our initial FOLifier implementation only handled fixed-arity relations.

META-SENTENCE indicates that a sentence occurs as a term (e.g. within a modal, or an exception).

META-VARIABLE indicates that the assertion references a meta-variable, which is an intrinsically higher-order language feature.

FUNCTION ARG CONSTRAINT indicates a constraint on the argument of a function. Our FOLifier implementation does not currently handle these.

HOOK indicates that the assertion contains a "hook" into some procedural code.

FUNCTION QUANTIFICATION, COLLECTION QUANTIFICATION and PREDICATE QUANTIFICATION indicate that the assertion quantified over a function, a collection (which translates into a predicate), or a predicate that was not a rule macro predicate.

INTER-MT indicates that the assertion involved some kind of inter-contextual reasoning between microtheories using the `ist` predicate that could not be translated into FOL.

SEQUENCE-VAR indicates that the assertion used a variable that stands for a sequence of terms rather than a single term. These are used in Cyc to write higher-order rules that quantify over predicates that could have varying arities.

NON-COLLECTION, NON-FUNCTION, NON-PREDICATE, and UNEXPECTED were cases which our FOLifier was unable to handle. Some of these were due to `Lambda` and `Kappa`, since function-denoting functions and predicate-denoting functions are inherently non-firstorderizable.

ILL-FORMED illustrates some kind of syntactic or semantic problem with the Cyc assertion itself.

4 The Cyc Inference Engine

The Cyc Inference Engine is a higher-order theorem prover optimized for the Cyc Knowledge Base. It differs from most automated theorem provers in several fundamental ways. Cyc's inference engine is designed to reason over a very large common-sense KB. It is incomplete, non-monotonic, and handles context reasoning natively. It is highly modular; in addition to modules implementing inference rules, it has modules implementing meta-reasoning and meta-meta-reasoning. It can dynamically handle theory revisions requiring only limited, local knowledge recompilation, and truth maintenance is always on.

4.1 Large Common-Sense KB

ResearchCyc contains over a million axioms, so the inference engine's datastructures and heuristics are designed to handle hundreds of thousands of constants and tens of thousands of predicates efficiently. Furthermore, the knowledge in Cyc's KB is *common-sense* knowledge. Common-sense knowledge is more often used in relatively shallow, "needle in a haystack" types of proofs than in deep mathematics-style proofs. Common-sense knowledge is more often used for constructive proofs than proofs by contradiction.

These factors have significantly influenced the design of Cyc's inference engine in ways quite different from most other automated theorem provers, which are often optimized for very different types of knowledge, such as formal verification, mathematics, or a single specific domain.

4.2 Incompleteness

Common-sense knowledge is tightly interconnected, and that fact combined with a very large KB make for intractably large branching factors in inference. Cyc's inference engine is heuristically guided and resource-bounded; it trades completeness for efficiency. At some point, the size and density of a KB become so large that any complete inference algorithm would be so slow as to be pragmatically useless. At that point, it becomes wise to give up completeness in favor of efficiency.

4.3 Highly Modular

Cyc, like most state of the art theorem provers, has a sophisticated set of heuristics used to rank potential inferences in a preference order. Also like most theorem provers, Cyc considers applying a set of inference rules at each inference step.

However, most theorem provers have a small handful of inference rules, usually including hyperresolution and paramodulation. In contrast, Cyc employs a "pandemonium" model in which hundreds to thousands of "Heuristic-Level (HL) modules", each of which implements an inference rule, check whether they are applicable to the given subproblem and make a bid to solve it. HL modules can be dynamically added or removed from the system without requiring any sort of retuning or recompilation.

4.4 Higher-Order

Some higher-order features are efficiently implemented via HL modules. For example, Cyc has an HL module that applies only to transitive predicates. When invoked, it performs a graph walk over the indexing structures of the KB, which is considerably more efficient than performing unrestricted resolution.

By representing the knowledge that a given predicate P is transitive as `TransitivePred(P)` rather than as a rule, Cyc gains parsimony in knowledge representation as well as efficiency in inference. If there are N transitive predicates in the universe of discourse, Cyc can represent this knowledge in $N + 1$ clauses: 1 clause per predicate plus 1 very general higher-order rule. In a straightforward FOL representation, $3N$ clauses would be required: 1 three-clause rule to express the transitivity of each predicate.

Many other higher-order features are handled analogously, with corresponding gains in both inference effiency and KR parsimony.

4.5 Contexts

Reasoning within a hierarchy of contexts is built into the innermost loop of Cyc's inference engine. Cyc maintains a dynamic contextual scope which can differ for each literal of each subproblem of an inference. Each axiom considered for use in a proof is first checked for relevance to the contextual scope. In the general case, this is done via efficient graph walking of the context hierarchy (a directed graph, not necessarily acyclic), and for the common cases, the computation is cached. Hence, Cyc can handle contextual and inter-contextual reasoning very efficiently.

5 Experimental Results (Preliminary)

Here we present the results of our preliminary experiments on performing inference with the first-orderized ResearchCyc KB.

In the last 20 years a number of efficient first-order theorem provers have been developed for doing sound and complete reasoning in first-order logic. The most common and successful design for these theorem provers is a resolution-style saturation strategy with sophisticated heuristics to determine clause and literal weights.

The gold standard for these theorem provers has been the TPTP (Thousands of Problems for Theorem Provers) library, which contains first-order encodings of problems from various domains as diverse as lattice theory to circuit verification. The challenge in most of these problems has been to find "deep inferences": starting from a relatively small set of axioms (small as compared to, say, Cyc's common-sense ontology) and returning a proof that requires a large number of inference steps.

Common-sense queries, on the other hand, typically require "shallow inferences", i.e. proofs that use a very small percentage of the entire KB and do not have many steps. The problem is to choose at each stage the correct line of inference to pursue out of all the possibilities. It has been observed (Reif & Schellhorn 1997) that goal-directed strategies are the only practical methods to use when the numbers of axioms are more than a few hundred.

Our ResearchCyc FOLification was motivated in part by the opportunity to experiment with first-order theorem proving strategies on the ResearchCyc KB and compare those results with the performance of Cyc's own inference engine.

Unfortunately we ran into a number of practical difficulties in using first-order theorem provers for our commonsense KB, a task for which they were not optimized. At the time of publication, we were unable to obtain the kind of extensive results that would enable us to draw confident conclusions. We instead present some initial work and our speculations.

With the exception of E, none of the theorem provers we tried were able to load more than 20% of the ResearchCyc KB without failing due to memory errors. In most cases, this was due to internal data structure limitations in the theorem-proving programs, which we are trying to fix with the help of the maintainers of these programs. Meanwhile, we have performed a few experiments with a subset of the ResearchCyc KB, as a feasibility study.

These experiments were performed as follows: We selected a set of 8 common-sense queries from an existing test corpus of queries for the ResearchCyc KB. This fixed set of queries was chosen before running any experiments. We selected these 8 in particular because they represent a diverse collection of different types of common-sense queries representable in ResearchCyc. None of these 8 queries turned out to be inherently dependent on higher-order reasoning; all of them were answerable by pure first-order reasoning on the translated KB.

We then randomly generated a subset of the FOLified ResearchCyc KB[2] containing less than 10% of the total number of axioms (about 100,000 out of 1,250,000) and manually ensured that the all the axioms needed to prove all the

[2] In the case of SPASS however, because of licensing issues, this was done on the OpenCyc KB, with virtually identical results.

Query	**VAMPIRE** (sec.)	**SPASS** (sec.)	**E** (sec.)	**Cyc** (sec.)
`(isa isa Individual)`	18.1	16.4	26	0.006
`(implies (and (subOrganizations ?Z ?X) (hasMembers ?X ?Y)) (hasMembers ?Z ?Y))`	3.4	5.2	12	2.5
`(typePrimaryFunction Bathtub TakingABath deviceUsed)`	1.6	2.8	9	0.02
`(typeBehaviorIncapable Doughnut Talking doneBy)`	43.4	29.1	132	0.03
`(implies (and (parts ?X ?Y) (parts ?Y ?Z)) (parts ?X ?Z))`	8.7	12.2	23	0.6
`(disjointWith Baseball-Ball Cube)`	176.5	343.0	1491	0.02
`(disjointWith HumanInfant Doctor-Medical)`	847.6	1239.1	2934	0.04
`(implies (and (isa ?CUP CoffeeCup) (isa ?COFFEE Coffee-Hot) (in-ContOpen ?COFFEE ?CUP)) (orientation ?CUP RightSideUp))`	218.4	462.5	574	0.7

Table 2: Inference Experiments

queries were present. (This manual step explains why we could only experiment with 8 queries.)

Finally we ran the theorem provers VAMPIRE, SPASS and E (which collectively represent the state of the art among ATP systems) on the KB + negated query. For SPASS, we had to first convert our KB into its native DFG format (Weidenbach 1999). The results are shown in Table 2. We have also shown the performance of the Cyc inference engine on these queries. The experiments were performed on an Athlon 2800 with 512 MB of RAM.

As an example of the completely different approaches to inference taken by the ATP systems and the Cyc inference engine, consider their behavior on the sixth query, `(disjointWith Baseball-Ball Cube)`.

Cyc solves the query in the following way: The inference engine delegates the problem to an HL module that can efficiently solve `disjointWith` queries. First it walks up the `genls` graph for `Cube`, marking each node as a goal. Then it walks up the `genls` graph for `Baseball-Ball`, and for each more general collection, it walks across all explicit `disjointWith` assertions to check whether it hits a goal node. In this particular example, the inference path is:

1. Mark `Cube`, `RectangularParallelepiped`, `Parallelepiped`, `Polyhedron`, etc. as goals
2. Iterate through each genl G of `Baseball-Ball`, e.g. `Ball`.
3. Iterate through each of G's asserted disjoins D, e.g. `Polyhedron`.
4. Check to see if D is a goal node.

This simplified walkthrough glosses over many details, not the least of which is the fact that microtheory reasoning is built into the innermost loop of this algorithm; on each traversal of each link in the `genls` and `disjointWith` graphs, Cyc does another graph walk of the `genlMt` graph, the graph in which Cyc's hierarchy of contexts is stored, to ensure that the link is relevant to the current context.

In comparison, the first-order approach is simple and uniform: Each `genls` and `disjointWith` assertion is translated into a rule such as $\forall x \forall y Baseball\text{-}Ball(x) \Rightarrow Ball(x)$ or $\neg(Ball(x) \wedge Polyhedron(x))$. Negating the query, $\exists y(Baseball\text{-}Ball(y) \wedge Cube(y))$ and running a resolution-style proof search will then yield a contradiction and hence a positive answer.

5.1 Interpretation

A few caveats have to be made before conclusions can be drawn from these experiments. In order to obtain these results, a number of simplifications were required in our experimental procedure. First, we had to scale down the size of the KB as described above in order to load it into all the theorem provers, whereas Cyc's inference engine was run on an unmodified KB (over 10 times the size of that used by the theorem provers). Furthermore, for all the queries except the last one, we 'flattened' the query context for the theorem provers, i.e. the query was asked in the BaseKB and all its supporting axioms were placed in the BaseKB. This in effect meant that first-order inference required one less unit propagation step on the `mtVisible` literal. The last query was not flattened, but was asked in the context of `TerrestrialFrameOfReferenceMt`. This may be a factor in its relative slowness. Cyc's inference engine did not flatten any of its queries or any of its KB. In all the deviations we made in the testing conditions for the FOL theorem provers versus the ResearchCyc inference engine, we ensured that we erred on the side of the theorem provers, i.e. the changes sped up their running times.

The immediate observation is that Cyc's performance is orders of magnitude better on almost all the queries. We believe that this validates our hypothesis that a specialized inference engine working natively with compact higher-order representations and special-purpose reasoning modules will perform better on common-sense queries than general first-order theorem proving strategies. However, much work remains to be done before this claim can be asserted with confidence.

The only query for which Cyc's result is within an order of magnitude of any of the other results is the second one, the $hasMembers$ query. This is because Cyc solves the query by hypothesizing terms for $?X$, $?Y$, and $?Z$ and performing forward inference on them before moving on to query-focused backward inference. 2.3 of the 2.5 seconds are spent in forward inference, which proves to be unnecessary for this particular query. This suggests further optimizations to Cyc; e.g. if the forward inference were done lazily rather than eagerly, Cyc could answer the question in 0.2 seconds rather than 2.5 seconds.

An alternative explanation for the difference in results is that our tests with most theorem provers were not performed under ideal settings. We did not have a chance to experiment extensively with the various parameter settings and heuristics that the programs offered; we used a uniform Set-Of-Support resolution setting for all of them. It could be that with, for example, an improved clause/literal weighting strategy, these theorem provers would become competitive with Cyc.

Another possible explanation for the difference in results is the inherent incompleteness of the FOLification procedure, as discussed in section 3. Perhaps it omits some axioms that are key to proving the queries efficiently in FOL, or perhaps it could translate to a form more amenable to other theorem provers. We hypothesize that this is not the case, and we plan to test our hypothesis by gathering input from the authors of other theorem provers, refining our FOLification procedure, and rerunning the experiment.

It should be noted, however, that most state of the art theorem provers have been optimized for mathematical problems requiring deep inference, such as those in the TPTP library. Cyc would almost certainly perform very poorly on those problems. Conversely, the FOLified ResearchCyc KB can be viewed as a new challenge problem for Automated Theorem Proving. With our results as a baseline, can theorem-proving strategies be found that improve performance on common-sense queries to match those of a specialized inference engine like Cyc?

The apparent poor performance of E relative to the other two theorem provers is surprising, especially given the fact that E performed very well in recent ATP competitions (Pelletier, Sutcliffe, & Suttner 2002) and that E was the only program that could load the entire ResearchCyc KB. This might be explained by the fact that E does not directly do Set-Of-Support resolution, but instead uses a clause weighting strategy that simulates it.

6 Conclusions and Future Work

The ResearchCyc FOLification tool was motivated by a number of goals. First, we aimed to measure the inherent higher-orderness of the ResearchCyc KB, which is a good indication of how much higher-orderness common-sense KBs in general can be expected to require. Going by sheer number, we were able to translate around 90% of the axioms. However, the other 10% of axioms that remained untranslated may form the core of Cyc's ontology. To verify this, experiments need to be set up with a large corpus of queries from ResearchCyc's test suite. The percentage of queries that can be answered by both ResearchCyc and any ATP system gives a measure of how much of the information content of the KB is actually translated, and how much of the inferential power actually carries over to FOL.

The second goal was to measure the performance of the Cyc inference engine against state of the art theorem provers. Our tentative conclusions are given in Section 5.1. The benefits of such a comparison could be mutual. First, if it is found that on certain classes of queries, there are theorem-proving strategies that work better than Cyc's inference engine, then we would like to explore the possibility of incorporating these methods into ResearchCyc, perhaps as HL modules. We are encouraged in this direction by the success of SUMO (Niles & Pease 2001) in using VAMPIRE.

Conversely, common-sense queries over large KBs of this scale represent an entirely new class of problems that have not been studied extensively in the automated reasoning community, mainly due to a lack of problem sets. The FOLified ResearchCyc KB could help kickstart research into this area. We have discussed with the maintainers of the TPTP problem library the possibility of generating a suite of common-sense benchmark problems from the ResearchCyc KB.

As for our experiments, a lot of work remains to be done before a completely fair comparison can be made between ResearchCyc and the ATP systems. Once we are able to perform experiments on the entire FOLified KB, then we can automatically run them on hundreds of queries in the ResearchCyc Test Corpus. The statistics gathered can then be used to draw definite conclusions about both representability and efficiency in reasoning.

It is quite likely that on many of the problems that modern theorem provers excel at (for example a theorem from group theory), Cyc's inference engine would perform poorly or not return an answer at all. Given the radically different design decisions of Cyc's inference engine versus those of most other theorem provers, it is not surprising that they have very different performance characteristics on different types of problems. Now that ResearchCyc is available to the academic community, these design decisions can finally be put to the proving ground. Is it possible to design a first-order theorem prover that can efficiently answer realistic common-sense queries over a large KB? Could such a theorem prover already exist, modulo a (perhaps more sophisticated) HOL to FOL translation mechanism? The experimental results presented in this paper are a first step toward answering these questions, and we challenge and encourage others to continue this work.

7 Acknowledgements

We would like to thank a number of people for their past and ongoing help. The first author would like to thank Eyal Amir for his encouragement and advice. We would like to thank the designers and maintainers of the theorem provers we used in our experiments, for helping us use their software on our KB: Stephen Schulz (E), Andrei Voronkov (VAMPIRE) and Thomas Hillenbrand (SPASS). We would especially like to thank Geoff Sutcliffe for his help with the TPTP translation and for running the model checking software Paradox on OpenCyc. This work was funded in part by ARDA's NIMD program.

References

Boolos, G. 1984. To be is to be the value of a variable (or to be some values of some variables). *Journal of Philosophy* 81:430–449.

Buvac, S. 1996. Quantificational logic of context. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*.

Giunchiglia, F. 1993. Contextual reasoning. *Epistemologia* XVI:345–364.

Lenat, D., and Guha, R. 1990. *Building Large Knowledge Based Systems*. Reading, Mass.: Addison Wesley.

McCarthy, J. 1986. Notes on formalizing contexts. In Kehler, T., and Rosenschein, S., eds., *Proceedings of the Fifth National Conference on Artificial Intelligence*, 555–560. Los Altos, California: Morgan Kaufmann.

Niles, I., and Pease, A. 2001. Towards a standard upper ontology. In Welty, C., and Smith, B., eds., *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems(FOIS-2001)*.

Pelletier, F.; Sutcliffe, G.; and Suttner, C. 2002. The Development of CASC. *AI Communications* 15(2-3):79–90.

Reif, W., and Schellhorn, G. 1997. Theorem proving in large theories. In Bonacina, M. P., and Furbach, U., eds., *Int. Workshop on First-Order Theorem Proving, FTP'97*, 119–124. Johannes Kepler Universität, Linz (Austria).

Riazanov, A. 2003. *Implementing an Efficient Theorem Prover*. Ph.D. Dissertation, University of Manchester.

Schulz, S. 2002. E - a brainiac theorem prover. *Journal of AI Communications* 15(2):111–126.

Sutcliffe, G., and Suttner, C. 1998. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning* 21(2):177–203.

Weidenbach, C. 1999. *Handbook of Automated Reasoning*. Elsevier. chapter SPASS: Combining superposition, Sorts and Splitting.